BIAX Corporation v. Intel
Civil Action No. 2:05-cv-184-TJW

EXHIBIT 1
(PART 4)

processor elements **640** according to the illustrated embodiment are identical. It is to be expressly understood, that any prior art type of processor element such as a micro-processor or other pipeline architecture could not be used under the teachings of the present invention, because such processors retain substantial state information of the program they are processing. However, such a processor could be programmed with software to emulate or simulate the type of processor necessary for the present invention.

The design of the processor element is determined by the instruction set architecture generated by the TOLL software and, therefore, from a conceptual viewpoint, is the most implementation dependent portion of this invention. In the illustrated embodiment shown in FIG. **21**, each processor element pipeline operates autonomously of the other processor elements in the system. Each processor element is homogeneous and is capable, by itself, of executing all computational and data memory accessing instructions. In making computational executions, transfers are from register to register and for memory interface instructions, the transfers are from memory to registers or from registers to memory.

Referring to FIG. **21**, the four-stage pipeline for the processor element **640** of the illustrated embodiment includes four discrete instruction registers **2100**, **2110**, **2120**, and **2130**. Each processor element also includes four stages: stage **1**, **2140**; stage **2**, **2150**; stage **3**, **2160**, and stage **4**, **2170**. The first instruction register **2100** is connected through the network **650** to the PIQ processor assignment circuit **1570** and receives that information over bus **2102**. The instruction register **2100** then controls the operation of stage **1** which includes the hardware functions of instruction decode and register **0** fetch and register **1** fetch. The first stage **2140** is interconnected to the instruction register over lines **2104** and to the second instruction register **2110** over lines **2142**. The first stage **2140** is also connected over a bus **2144** to the second stage **2150**. Register **0** fetch and register **1** fetch of stage **1** are connected over lines **2146** and **2148**, respectively, to network **670** for access to the register file **660**.

The second instruction register **2110** is further interconnected to the third instruction register **2120** over lines **2112** and to the second stage **2150** over lines **2114**. The second stage **2150** is also connected over a bus **2152** to the third stage **2160** and further has the memory write (MEM WRITE) register fetch hardware interconnected over lines **2154** to network **670** for access to the register file **660** and its condition code (CC) hardware connected over lines **2156** through network **670** to the condition code storage of context file **660**.

The third instruction register **2120** is interconnected over lines **2122** to the fourth instruction register **2130** and is also connected over lines **2124** to the third stage **2160**. The third stage **2160** is connected over a bus **2162** to the fourth stage **2170** and is further interconnected over lines **2164** through network **650** to the data cache interconnection network **1590**.

Finally, the fourth instruction register **2130** is interconnected over lines **2132** to the fourth stage, and the fourth stage has its store hardware (STORE) output connected over lines **2172** and its effective address update (EFF. ADD.) hardware circuit connected over lines **2174** to network **670** for access to the register file **660**. In addition, the fourth stage has its condition code store (CC STORE) hardware connected over lines **2176** through network **670** to the condition code storage of context file **660**.

The operation of the four-stage pipeline shown in FIG. **21** will now be discussed with respect to the example of Table 1 and the information contained in Table 19 which describes the operation of the processor element for each instruction.

TABLE 19

	Instruction I0, (I1):
Stage 1	Fetch Reg to form Mem-adx
Stage 2	Form Mem-adx
Stage 3	Perform Memory Read
Stage 4	Store R0, (R1)
	Instruction I2:
Stage 1	Fetch Reg R0 and R1
Stage 2	No-Op
Stage 3	Perform multiply
Stage 4	Store R2 and CC
	Instruction I3:
Stage 1	Fetch Reg R2 and R3
Stage 2	No-Op
Stage 3	Perform addition
Stage 4	Store R3 and CC
	Instruction I4:
Stage 1	Fetch Reg R4
Stage 2	No-Op
Stage 3	Perform decrement
Stage 4	Store R4 and CC

For instructions **I0** and **I1**, the performance by the processor element **640** in FIG. **21** is the same except in stage **4**. The first stage is to fetch the memory address from the register which contains the address in the register file. Hence, stage **1** interconnects circuitry **2140** over lines **2146** through network **670** to that register and downloads it into register **0** from the interface of stage **1**. Next, the address is delivered over bus **2144** to stage **2**, and the memory write hardware forms the memory address. The memory address is then delivered over bus **2152** to the third stage which reads memory over **2164** through network **650** to the data cache interconnection network **1590**. The results of the read operation are then stored and delivered to stage **4** for storage in register **R0**. Stage **4** delivers the data over lines **2172** through network **670** to register **R0** in the register file. The same operation takes place for instruction **I1** except that the results are stored in register **1**. Hence, the four stages of the pipeline (Fetch, Form Memory Address, Perform Memory Read, and Store The Results) flow data through the pipe in the manner discussed, and when instruction **I0** has passed through stage **1**, the first stage of instruction **I1** commences. This overlapping or pipelining is conventional in the art.

Instruction **I2** fetches the information stored in registers **R0** and **R1** in the register file **660** and delivers them into registers **REG0** and **REG1** of stage **1**. The contents are delivered over bus **2144** through stage **2** as a no operation and then over bus **2152** into stage **3**. A multiply occurs with the contents of the two registers, the results are delivered over bus **2162** into stage **4** which then stores the results over lines **2172** through network **670** into register **R2** of the register file **660**. In addition, the condition code data is stored over lines **2176** in the condition code storage of context files **660**.

Instruction **I3** performs the addition of the data in registers **R2** and **R3** in the same fashion, to store the results, at stage **4**, in register **R3** and to update the condition code data for that instruction. Finally, instruction **I4** operates in the same fashion except that stage **3** performs a decrement of the contents of register **R4**.

Hence, according to the example of Table I, the instructions for PEO, would be delivered from the PIQO in the following order: IO, I2, and I3. These instructions would be sent through the PEO pipeline stages (S1, S2, S3, and S4), based the upon the instruction firing times (T16, T17, and T18), as follows:

TABLE 20

PE	Inst	T16	T17	T18	T19	T20	T21
PE0:	10	S1	S2	S3	S4		
	I2		S1	S2	S3	S4	
	I3			S1	S2	S3	S4
PE1:	I1	S1	S2	S3	S4		
PE2:	I4	S1	S2	S3	S4		

The schedule illustrated in Table 20 is not however possible unless data chaining is introduced within the pipeline processor (intraprocessor data chaining) as well as between pipeline processors (interprocessor data chaining). The requirement for data chaining occurs because an instruction no longer completely executes within a single time cycle illustrated by, for example, instruction firing time T16. Thus, for a pipeline processor, the TOLL software must recognize that the results of the store which occurs at stage 4 (T19) of instructions I0 and I1 are needed to perform the multiply at stage 3 (T19) of instruction I2, and that fetching of those operands normally takes place at stage 1 (T17) of instruction I2. Accordingly, in the normal operation of the pipeline, for processors PE0 and PE1, the operand data from registers R0 and R1 is not available until the end of firing time T18 while it is needed by stage 1 of instruction I2 at time T17.

To operate according to the schedule illustrated in Table 20, additional data (chaining) paths must be made available to the processors, paths which exist both internal to the processors and between processors. These paths, well known to those practiced in the art, are the data chaining paths. They are represented, in FIG. 21, as dashed lines 2180 and 2182. Accordingly, therefore, the resolution of data dependencies between instructions and all scheduling of processor resources which are performed by the TOLL software prior to program execution, take into account the availability of data chaining when needed to make available data directly from the output, for example, of one stage of the same processor or a stage of a different processor. This data chaining capability is well known to those practiced in the art and can be implemented easily in the TOLL software analysis by recognizing each stage of the pipeline processor as being, in effect, a separate processor having resource requirements and certain dependencies, that is, that an instruction when started through a pipeline will preferably continue in that same pipeline through all of its processing stages. With this in mind, the speed up in processing can be observed in Table 20 where the three machine cycle times for the basic block are completed in a time of only six pipeline cycles. It should be borne in mind that the cycle time for a pipeline is approximately one-fourth the cycle time for the non-pipeline processor in the illustrated embodiment of the invention.

The pipeline of FIG. 21 is composed of four equal (temporal) length stages. The first stage 2140 performs the instruction decode, determines what registers to fetch and store, and performs up to two source register fetches which can be required for the execution of the instruction.

The second stage 2150 is used by the computational instructions for the condition code fetch if required. It is also the effective address generation stage for the memory interface instructions.

The effective address operations that are supported in the preferred embodiment of the invention are:

1. Absolute address

The full memory address is contained in the instruction.

2. Register indirect

The full memory address is contained in a register.

3. Register indexed/based

The full memory address is formed by combining the designated registers and immediate data.

a. Rn op K
b. Rn op Rm
c. Rn op K op Rm
d. Rn op Rm op K

where "op" can be addition (+), subtraction (−), or multiplication (*) and "K" is a constant.

As an example, the addressing constructs presented in the matrix multiply inner loop example are formed from case 3-a where the constant "K" is the length of a data element within the array and the operation is addition (+).

At a conceptual level, the effective addressing portion of a memory access instruction is composed of three basic functions; the designation and procurement of the registers and immediate data needed for the calculation, the combination of these operands in order to form the desired address, and if necessary, updating of any one of the registers involved. This functionality is common in the prior art and is illustrated by the autoincrement and autodecrement mode of addressing available in the DEC processor architecture. See, for example, DEC VAX Architecture Handbook.

Aside from the obvious hardware support required, the effective addressing is supported by the TOLL software, and impacts the TOLL software by adding functionality to the memory accessing instructions. In other words, an effective address memory access can be interpreted as a concatenation of to operations, the first being the effective address calculation and the second being the actual memory access. This functionality can be easily encoded into the TOLL software by one skilled in the art in much the same manner as an add, subtract or multiply instruction would be.

The described effective addressing constructs are to be interpreted as but one possible embodiment of a memory accessing system. There are a plethora of other methods and modes for generating a memory address that are known to those skilled in the art. In other words, the effective addressing constructs described above are for design completeness only, and are not to be construed as a key element in the design of the system.

Referring to FIG. 22, various structures of data or data fields within the pipeline processor element of FIG. 21 are illustrated for a system which is a multi-user system in both time and space. As a result, across the multiple pipelines, instructions from different users may be executing, each with its own processor state. As the processor state is not typically associated with the processor element, the instruction must carry along the identifiers that specify this state. This processor state is supported by the LRD, register file and condition code file assigned to the user.

A sufficient amount of information must be associated with each instruction so that each memory access, condition code access or register access can uniquely identify the target of the access. In the case of the registers and condition codes, this additional information constitutes the absolute value of the procedural level (PL) and context identifiers (CI) and is attached to the instruction by the SCSM attachment unit **1650**. This is illustrated in FIGS. **22***a*, **22***b*, and **22***c* respectively. The context identifier portion is used to determine which register or condition code plane (FIG. **6**) is being accessed. The procedural level is used to determine which procedural level of registers (FIG. **13**) is to be accessed.

Memory accesses also require that the LRD that supports the current user be identified so that the appropriate data cache can be accessed. This is accomplished through the context identifier. The data cache access further requires that a process identifier (PID) for the current user be available to verify that the data present in the cache is indeed the data desired. Thus, an address issued to the data cache takes the form of FIG. **22***d*. The miscellaneous field is composed of additional information describing the access, for example, read or write, user or system, etc.

Finally, due to the fact that there can be several users executing across the pipelines during a single time interval, information that controls the execution of the instructions, and which would normally be stored within the pipeline, must be associated with each instruction instead. This information is reflected in the ISW field of an instruction word as illustrated in FIG. **22***a*. The information in this field is composed of control fields like error masks, floating point format descriptors, rounding mode descriptors, etc. Each instruction would have this field attached, but, obviously, may not require all the information. This information is used by the ALU stage **2160** of the processor element.

This instruction information relating to the ISW field, as well as the procedural level, context identification and process identifier, are attached dynamically by the SCSM attacher (**1650**) as the instruction is issued from the instruction cache.

Although the system of the present invention has been specifically set forth in the above disclosure, it is to be understood that modifications and variations can be made thereto which would still fall within the scope and coverage of the following claims.

We claim:

1. A parallel processor system for processing natural concurrencies in streams of low level instructions contained in a plurality of programs in said system, each of said streams having a plurality of single entry-single exit (SESE) basic blocks (BBs), said system comprising:

means (**160**) for statically adding intelligence to each instruction in each of said plurality of basic blocks for each said program, said added intelligence at least having a logical processor number (LPN) and an instruction firing time (IFT),

a plurality of context files (**660**), each of said context files being assigned to one of said plurality of programs for processing one of said programs, each of said context files having at least a plurality of registers and condition code storage for containing processing status information,

a plurality of logical resource drivers (LRDs), each logical resource driver being assigned to one of said plurality of context files, each of said logical resource drivers being receptive of said basic blocks corresponding to the program instruction stream of said assigned program from said adding means, each of said logical resource drivers comprising:

(a) a plurality of queues (**1730**), and

(b) means operative on said plurality of said basic blocks containing said intelligence from said adding means for delivering said instructions in each said basic block into said plurality of queues based on said logical processor number, said instructions in each said queue being entered according to said instruction firing time wherein the earliest instruction firing time is entered first,

a plurality of individual processor elements (PEs), each of said processor elements being free of any context information,

means (**650**) connecting said plurality of processor elements to said plurality of logical resource drivers for transferring said instruction with the earliest instruction firing time, first in said queues, from each of said logical resource drivers, in a predetermined order, to individually assigned processor elements, each said processor element having means for processing said transferred instruction,

first means (**670**) for connecting each of said processor elements with any one of said plurality of context files, each of said processor elements having means for accessing any of a plurality of registers and condition code storage in a program's context file during the processing of the program's instruction,

a plurality of memory locations (**610**), and

second means including said logical resource drivers for connecting each of said processor elements with any one of said plurality of memory locations, each said processor element having means for accessing said memory locations during said processing of each said instruction.

2. A parallel processor system for processing natural concurrence in streams of low level instructions contained in a plurality of programs in said system, each of said streams having a plurality of single entry-single exit (SESE) basic blocks (BBs), said system comprising

means (**160**) for statically adding intelligence to each instruction in each of said plurality of basic blocks for each of said programs, said added intelligence representing at least a logical processor number (LPN) and an instruction firing time (IFT),

a plurality of context files (**660**), each of said context files being assigned to one of said plurality of programs for processing one of said programs, each of said context files having at least a plurality of registers and a condition code storage for containing processing status information,

a plurality of logical resource drivers (LRDs), each logical resource driver being assigned to one of said plurality of context files, each of said logical resource drivers being receptive of said basic blocks corresponding to the program instruction stream of said assigned program from said adding means, each of said logical resource drivers comprising:

(a) a plurality of queues (**1730**), and

(b) means operative on said plurality of said basic blocks containing said intelligence from said adding means for delivering said instructions in each said basic block into said plurality of queues based on said logical processor number, said

instructions in each said queue being entered according to said instruction firing time wherein the earliest instruction firing time is entered first,

a plurality of context free individual processor elements (PEs),

means (**650**) connecting said plurality of processor elements to said plurality of logical resource drivers for transferring said instructions from each of said logical resource drivers, in a predetermined order, to individually assigned processor elements, each said processor element having means for processing said transferred instruction,

first means (**670**) for connecting each of said processor elements with any one of said plurality of context files, each of said processor elements having means for accessing any of a plurality of registers and condition code storages in a program's context file during said processing of the program's instruction,

a plurality of memory locations (**610**), and

second means including said logical resource drivers for connecting each of said processor elements with any one of said plurality of memory locations, each said processor element having means for accessing said memory locations during said processing of each said instruction.

3. A parallel processor system for processing natural concurrencies in streams of low level instructions contained in a plurality of programs in said system, each of said streams having a plurality of single entry-single exit (SESE) basic blocks (BBs), said system comprising:

means (**160**) for statically adding intelligence to each instruction in each of said plurality of basic blocks for each of said programs, said added intelligence representing at least a logical processor number (LPN) and an instruction firing time (IFT),

a plurality of context files (**660**), each of said context files being assigned to one of said plurality of programs for processing one of said programs, each of said context files having a plurality of registers and a condition code storage for containing processing status information,

a plurality of logical resource drivers (LRDs), each logical resource driver being assigned to one of said plurality of context files, each of said logical resource drivers being receptive of said basic blocks corresponding to the program instruction stream of said assigned program from said adding means for storing said instructions, and fetching, in order of said instruction firing time, said instructions in each basic block, and delivering said instructions according to the logical processor number for each instruction,

a plurality of individual context free processor elements (PEs),

means (**650**) connecting said plurality of processor elements to said plurality of logical resource drivers for transferring said instructions from each of said logical resource drivers, in a predetermined order, to individually assigned processor elements, each said processor element having means for processing said transferred instruction,

first means (**670**) for connecting each of said processor elements with any one of said plurality of context files, each of said processor elements having means for accessing any of a plurality of registers and condition code storage in a program's context file during said processing of the program's instruction,

a plurality of memory locations (**610**), and

second means including said logical resource drivers for connecting each of said processor elements with any one of said plurality of memory locations, each said processor element having means for accessing said memory locations during said processing of each said instruction.

4. The parallel processor system according to claims **1**, **2**, or **3** in which:

said adding means further has means for statically adding shared context storage mapping (S-SCSM) information to said instructions, said statically added shared context storage information containing level information for each said instruction in order to identify the different program levels contained within each said program,

said context files having a different set of registers for each said program level,

said logical resource drivers dynamically adding shared context storage mapping information to said instructions in response to said statically added information for identifying subroutine levels of the program,

said dynamically added shared context storage mapping information corresponding to said sets of registers, and

said processor elements further having means for processing its instructions using sets of registers identified by said dynamically added shared context storage mapping information.

5. The parallel processor system according to claim **1**, **2**, or **3** in which:

each of said logical resource drivers further comprises means for dynamically adding shared context storage mapping (D-SCSM) information to each said instruction, said dynamically added shared context storage information containing the identity of the context file assigned to the program contained within each said logical resource driver,

each of said context files being assigned to one of said logical resource drivers, each said context file being identified by said dynamically added shared context storage mapping information, and

said processor elements further having means for processing each of its instructions in the context file identified by said dynamically added shared context storage mapping information.

6. A parallel processor system for processing natural concurrencies in streams of low level instructions contained in a plurality of programs in said system, each of said streams having a plurality of single entry-single exit (SESE) basic blocks (BBs), said system comprising:

means (**160**) for adding intelligence to each instruction in each of said plurality of basic blocks, said added intelligence representing at least a logical processor number (LPN) and an instruction firing time (IFT),

a plurality of context files (**660**), each of said context files being assigned to one of said plurality of programs, each of said context files having a plurality of register resources,

a plurality of logical resource drivers (LRDs), each logical resource driver being assigned to one of said plurality of context files, each of said logical resource drivers being receptive of said basic blocks corresponding to the instruction stream of

said assigned program from said adding means for storing said instructions in each basic block according to the logical processor number for each instruction,

a plurality of individual context free processor elements (PEs),

means (**650**) connecting said plurality of processor elements to said plurality of logical resource drivers for transferring said instructions from each of said logical resource drivers, in a predetermined order according to the instruction firing time, to individually assigned processor elements, each said processor element having means for processing said transferred instruction,

first means (**670**) for connecting each of said processor elements with any one of said plurality of context files, each said processor element having means for accessing said resources in a program's context file during said processing of the program's instruction,

a plurality of memory locations (**610**), and

second means including said logical resource drivers for connecting each of said processor elements with any one of said plurality of memory locations, each said processor element having means for accessing said memory locations during said processing of each said instruction.

7. A parallel processor system for processing natural concurrencies in streams of low level instructions contained in a plurality of programs in said system, each of said streams having a plurality of single entry-single exit (SESE) basic blocks (BBs), said system comprising:

means (**160**) for adding intelligence to each instruction in each of said plurality of basic blocks,

a plurality of context files (**660**), each of said context files being assigned to at least one of said plurality of programs each of said context files having a plurality of register resources,

a plurality of logical resource drivers (LRDs) with each logical resource driver being assigned to one of said plurality of context files, each of said logical resource drivers being receptive of said basic blocks corresponding to the program instruction stream of said at least one assigned program from said adding means for storing said instructions in each basic block,

a plurality of individual processor elements (PEs),

means (**650**) connecting said plurality of processor elements to said plurality of logical resource drivers for transferring said instructions from each of said logical resource driver to individually assigned processor elements in accordance with said added intelligence added to the instructions,

first means (**670**) for connecting each of said processor elements to any one of said plurality of context files, each said processor element having means for accessing any resource in a program's context file during said processing of the program's instruction,

a plurality of memory locations (**610**), and

second means including said logical resource drivers for connecting each of said processor elements with any one of said plurality of memory locations, each said processor element having means for accessing said memory locations during said processing of each said instruction.

8. The parallel processor system according to claims **6** or **7** in which:

said adding means further has means for adding to each said instruction, information containing level information for each said instruction for identifying different program levels contained within each said program,

said context files have a different set of register resources for each said program level,

each said set of resources is identified by said added information, and

said processor elements further have means for processing each of its instructions in a set of register resources identified by said added information.

9. The parallel processor system according to claims **6** or **7** in which

each of said logical resource drivers further comprises means for adding information to each said instruction, said added information containing the identity of the context file assigned to the programs contained within each said logical resource driver,

each of said context files is assigned to one of said logical resource drivers, each said context file being identified by said added information, and

said processor elements further have means for processing each of its instructions using the context file identified by said added information.

10. A parallel processor system for processing natural concurrencies in streams of low level instructions contained in a plurality of programs in said system, at least one of said programs having a plurality of different program levels, each of said streams having a plurality of single entry-single exit (SESE) basic blocks (BBs), said system comprising:

means for adding intelligence to each instruction in each of said plurality of basic blocks, said added intelligence containing program level information for each said instruction to identify the different program levels contained within each said program,

a plurality of context files (**660**), each of said context files being assigned at least one of said plurality of programs, each of said context files having a plurality of register resources with a different set of register resources for each said program level, each said set of resources being identified by said added intelligence,

a plurality of logical resource drivers (LRDs), each logical resource driver being assigned to one of said plurality of context files, each of said logical resource drivers being receptive of said basic blocks corresponding to the program instruction stream of said at least one assigned program from said adding means for storing said instructions of each basic block of the assigned program instruction stream, each of said logical resource drivers further having means for adding information to each said instruction, said added information containing the identity of the context file assigned to the programs contained within each said logical resource driver,

a plurality of individual processor elements (PEs),

means (**650**) connecting said plurality of processor elements to said plurality of logical resource drivers for transferring said instructions from each of said logical resource drivers to individually assigned processor elements,

first means (**670**) for connecting each of said processor elements to any one of said plurality of context files, each said processor element having means for

accessing a set of resources, as identified by said added intelligence, in a program's context file as identified by said added information, during said processing of the program's instruction,

a plurality of memory locations (**610**), and

second means including said logical resource drivers for connecting each of said processor elements with any one of said plurality of memory locations, each said processor element having means for accessing said memory locations during said processing of each said instruction.

11. A parallel processor system for processing natural concurrencies in streams of low level instructions contained in a program, each of said streams having a plurality of single entry-single exit (SESE) basic blocks (BBs), said system comprising:

means (**160**) for statically adding intelligence to each instruction in each of said plurality of basic blocks, said added intelligence representing at least a logical processor number (LPN) and an instruction firing time (IFT) for the instructions,

a logical resource driver (LRD) receptive of said basic blocks corresponding to the program instruction stream from said adding means for storing said instructions and said logical resource driver further having means for fetching, in order of said instruction firing time, said instructions in each basic block, and delivering said instructions according to the logical processor number for each instruction,

a plurality of individual context free processor elements (PEs),

means (**650**) connecting said plurality of processor elements with said logical resource driver for transferring said instructions from said logical resource driver to individually assigned processor elements, each said processor element having means for processing said transferred instruction,

a plurality of shared storage resources (**660**)

first means (**670**) for connecting each of said processor elements with any one of said plurality of resources, each of said processor elements having means for accessing any one of said resources during said instruction processing,

a plurality of memory locations (**610**), and

second means including said logical resource drivers for connecting each of said processor elements with any one of said plurality of memory locations, each said processor element having means for accessing said memory locations during said instruction processing.

12. A parallel processor system for processing natural concurrencies in streams of low level instructions contained in a program, each of said streams having a plurality of single entry-single exit (SESE) basic blocks (BBs), said system comprising:

means (**160**) for statically adding intelligence to each instruction in each of said plurality of basic blocks, said added intelligence representing at least a logical processor number (LPN) and an instruction firing time (IFT),

a logical resource driver (LRD) receptive of said basic blocks corresponding to the program instruction stream from said adding means for storing said instructions and said logical resource driver further having means for fetching, in order of said instruction firing time, said instructions in each basic block, and delivering said instructions according to the logical processor number for each instruction,

a plurality of individual processor elements (PEs),

first means (**650**) connecting said plurality of processor elements with said logical resource driver for transferring said instructions from said logical resource driver to individually assigned processor elements, each said processor element having means for processing said transferred instruction,

a plurality of shared storage resources (**660**), and

second means (**670**) for connecting each of said processor elements with any one of said plurality of shared storage resources, each of said processor elements having means for acessing any one of said shared storage resources during said instruction processing.

13. A parallel processor system for processing natural concurrencies in streams of low level instructions contained in a program, each of said streams having a plurality of single entry-single exit (SESE) basic blocks (BBs), said system comprising:

means (**160**) for statically adding intelligence to each instruction in each of said plurality of basic blocks,

a logical resource driver (LRD) receptive of said basic blocks corresponding to the program instruction stream from said adding means for storing said basic blocks,

a plurality of individual context free processor elements (PEs),

means (**650**) connecting said plurality of processor elements with said logical resource driver for transferring said instructions from said logical resource driver to individually assigned processor elements in accordance with the added intelligence in the instructions, each said processor element having means for processing said transferred instruction,

a plurality of shared storage resources (**660**), and

means (**670**) for connecting each of said processor elements with any one of said plurality of shared storage resources in accordance with the added intelligence in the instructions, each of said shared storage resources during said instruction processing.

14. The parallel processor system according to claims **11**, **12**, or **13** in which:

said adding means further has means for statically adding program level information to each said instruction to identify the different program levels contained within each said program,

said shared storage resources have a different set of resources for each said program level,

each said set of resources is identified by said statically added level information, and

each said processor element further has means for processing each of its instructions in a set of resources identified by said statically added level information.

15. A method for parallel processing in a plurality of processor elements natural concurrencies in streams of low level instructions contained in the programs of a plurality of users, each of the streams having a plurality of single entry-single exit (SESE) basic blocks (BBs), said method comprising the steps of:

statically adding intelligence representing the natural concurrencies existing within the instructions in each basic block of the programs, said step of adding for each program comprising the steps of:

(a) ascertaining the resource requirements of each instruction within each basic block to determine the natural concurrencies in each basic block,

(b) identifying logical resource dependencies between instructions,

(c) assigning condition code storage (CCs) to groups of resource dependent instructions, so that dependent instructions can execute on the same or different processor elements,

(d) determining the earliest possible instruction firing time (IFT) for each of said instructions in each of said plurality of basic blocks,

(e) adding said instruction firing times to each instruction in each of said plurality of basic blocks,

(f) assigning a logical processor number (LPN) to each instruction in each of said basic blocks,

(g) adding said logical processor numbers to each instruction in each of said basic blocks, and

(h) repeating steps (a) through (g) until all basic blocks are processed for each of said programs, and

processing the instruction having the statically added intelligence for the programs, the step of processing further comprising the steps of:

(i) delivering the instructions into logical resource drivers, each said program of a user being assigned to a different logical resource driver,

(j) selecting instructions from the logical resource drivers in a predetermined order based on the instruction firing time,

(k) storing the selected instructions in queues of the logical resource driver based on the logical processor number,

(l) generating dynamic shared context storage mapping (D-SCSM) information for each instruction,

(m) selectively connecting the queues of each logical resource driver to processor elements (PEs) said queues being connected in a predetermined order so that one instruction having the earliest instruction firing time from each queue is first delivered to a given processor element,

(n) processing said one instruction from each queue in each said connected processor element,

(o) obtaining the input data for processing said delivered instruction from shared storage locations identified by said instruction in a context file identified by said dynamic shared context storage mapping information,

(p) storing the results of said processing of said delivered instruction in shared storage identified by said dynamic information contained in said instruction, and

(q) repeating steps (i) through (p) until all instructions in each of said plurality of basic blocks for all said programs are processed.

16. The method of claim **15** wherein said step of statically adding intelligence further comprises the step of re-ordering said instructions in each of said basic blocks based upon said instruction firing times wherein the earliest firing times are listed first.

17. The method of claim **15** wherein said step of statically adding intelligence further comprises the step of adding static shared context storage mapping information (S-SCSM) to each instruction to identify the relative program levels associated with said instructions and wherein said step of obtaining input further comprises the step of obtaining said input data from a shared stored location procedural level identified at least in part by the aforesaid dynamically added information.

18. The method of claim **15** wherein said step of statically adding intelligence further comprises the step of adding static shared context storage mapping (S-SCSM) information to each instruction to identify the program level of said instruction.

19. A method for parallel processing natural concurrencies in streams of low level instructions contained in the programs of a plurality of users located in a system having a plurality of processor elements and shared storage locations, each of the streams having a plurality of single entry-single exit (SESE) basic blocks (BBs), said method comprising the steps of:

statically adding intelligence to the instructions in each basic block of the programs, said added intelligence identifying the natural concurrencies within each basic block, said added intelligence having a least an instruction firing time (IFT) and a logical processor number (LPN), and

processing the instructions having the statically added intelligence for executing the programs, the step of processing further comprising the steps of:

(a) delivering the instructions into the system, each said user being assigned to a different context file in said system,

(b) dynamically generating shared context state mapping (D-SCSM) information for each instruction identifying said context file containing shared storage locations,

(c) separately storing the delivered instructions in the system based on the logical processor number (LPN),

(d) selectively connecting the separately stored instructions to the processor elements assigned to the logical processor number for the instructions, said separately stored instructions being delivered in a predetermined order so that one instruction having the earliest instruction firing time from each of the separately stored instructions is delivered to a given processor element,

(e) processing said one instruction from each connected separately stored instructions in each said connected processor element,

(f) obtaining the input data for processing said connected instruction from a shared storage location identified at least in part by said shared context storage mapping information,

(g) storage the results of said processing of said connected instruction in a shared storage location identified in part by said shared context storage mapping information, and

(h) repeating steps (a) through (g) until all instructions of each of said plurality of basic blocks for all of said programs are processed.

20. A method for parallel processing, in a system, natural concurrencies in streams of low level instructions contained in a program, each of the streams having a plurality of single entry-single exit (SESE) basic blocks (BBs), said system having a plurality of processor elements and a plurality of shared storage locations, said method comprising the steps of:

statically adding intelligence representing the natural concurrencies existing within the instructions in each basic block, said step of adding comprising the steps of:

(a) ascertaining the resource requirements of each instruction within each basic block to determine the natural concurrencies in each basic block,

(b) identifying logical resource dependencies between instructions,

(c) assigning condition code storage (CCs) to groups of resource dependent instructions, such that dependent instructions can execute on the same of different processor elements,

(d) determining the earliest possible instruction firing time (IFT) for each of said instructions in each of said plurality of basic blocks,

(e) adding said instruction firing times to each instruction in each of said plurality of basic blocks,

(f) assigning a logical processor number (LPN) to each instruction in each of said basic blocks,

(g) adding said logical processor numbers to each instruction in each of said basic blocks, and

(h) repeating steps (a) through (g) until all basic blocks are processed for said program, and

processing the instructions having the statically added intelligence in the system, the step of processing further comprising the steps of:

(i) separately storing instructions delivered in the system based on the logical processor number,

(j) selectively connecting the separately stored instructions to the processor element (PE) assigned to the logical processor number, said separately stored instructions being delivered in a predetermined order so that one instruction having the earliest instruction firing time is connected to a given processor element,

(k) processing said one instruction from each of the separately stored instructions in each said connected processor element, and

(l) repeating steps (i) through (k) until all instructions in each of said plurality of basic blocks for said program are processed.

21. A method for parallel processing, in a system, natural concurrencies in streams of low level instructions contained in a program, each of the streams having a plurality of single entry-single exit (SESE) basic blocks (BBs), said system having a plurality of processor elements and a plurality of shared storage locations, said method comprising the steps of:

statically adding intelligence representing the natural concurrencies existing within the instructions in each basic block, said step of adding comprising the steps of:

(a) determining the earliest possible instruction firing time (IFT) for each of said instructions in each of said plurality of basic blocks,

(b) adding said instruction firing times to each instruction in each of plurality of basic blocks,

(c) assigning a logical processor number (LPN) to each instruction in each of said basic blocks, and

(d) adding said logical processor numbers to each instruction in each of said basic blocks, and

processing the instructions having the statically added intelligence, the step of processing further comprising the steps of:

(e) separately storing instructions delivered in the system based on the logical processor number,

(f) selectively connecting said separately stored instructions to processor elements (PEs) assigned to a given logical processor number, said instructions connected in a predetermined order so that one instruction having the earliest instruction firing time from each of said separately stored instructions is connected to a given processor element,

(g) processing said one connected instruction in each said connected processor element,

(h) obtaining the input data for processing said connected instruction from a shared storage location identified by said instruction,

(i) storing the results of said processing of said connected instruction in a shared storage location identified by said instruction, and

(j) repeating steps (e) through (i) until all instructions are processed in each of said plurality of basic blocks for said program.

22. A method for parallel processing natural concurrencies in a program in a system having a plurality of processor elements (PEs), said processor element having access to input data located in a plurality of shared resource locations, said program having a plurality of single entry-single exit (SESE) basic blocks (BBs) with each of said basic blocks (BBs) having a stream of instructions, said method comprising the steps of:

ascertaining the resource requirements of each instruction within each of said basic blocks,

identifying logical resource dependencies between instructions,

assigning condition code storage (CCs) to groups of resource dependent instructions, such that dependent instructions can execute on the same of different processor elements,

determining the earliest possible instruction firing time (IFT) for each of the instructions in said plurality of basic blocks,

adding said instruction firing times (IFTs) to each instruction in each of said plurality of basic blocks in response to said determination,

assigning a logical processor number (LPN) to each instruction in each of said basic blocks,

adding said assigned logical processor number (LPN) to each instruction in each of said plurality of basic blocks in response to said assignment,

separately storing the instructions, with said added instruction firing time and said added logical processor numbers, based on the logical processor number, each group of said separately stored instructions containing instructions having only the same logical processor number,

selectively connecting said separately stored instructions to said processor elements based on the logical processor number, and

each said processor element receiving the instruction in said connected group having the earliest instruction firing time first, said processor element having means for performing the steps of:

(a) obtaining input data for processing said received instruction from a shared storage location in said plurality of shared resource locations identified by said instruction,

(b) storing the results based upon the aforesaid step of processing in a shared storage location in said plurality of shared resource locations identified by said received instruction, and

(c) repeating the aforesaid steps (a) and (b) for the next received instruction until all instructions are processed.

23. The method of claim **22** further comprising the steps of forming execution set of basic blocks in response to said steps of adding said instruction firing times and logical processor numbers wherein branches from any given basic block within a given execution set

to a basic block in another execution set is statistically minimized.

24. The method of claim **22** further comprising the step of:

adding shared context storage mapping information to each instruction, and

wherein said step of processing comprises the step of processing each instruction requiring at least one set of shared resources, said at least one set being identified by said shared context storage mapping information, so each program routine can access, in addition to the routine's set of procedural level resources, at least one other set of resources.

25. A method for parallel processing instructions of a program having natural concurrencies with a plurality of processor elements (PEs), said processor elements having access to input data located in a plurality of shared resource locations, said program having a plurality of single entry-single exit (SESE) basic blocks (BBs) with each of said basic blocks (BBs) having a stream of method instructions, said method comprising the steps of:

ascertaining the resource requirements of each instruction within each of said basic blocks,

identifying logical resource dependencies between instructions,

assigning condition code storage (CCs) to groups of resource dependent instructions, such that dependent instructions can execute on the same or different processor elements,

determining the earliest possible instruction firing time (IFT) for each of the instructions in each of said plurality of basic blocks,

adding said instruction firing times to each instruction in each of said plurality of basic blocks in response to said determination,

assigning a logical processor number (LPN) to each instruction in each of said basic blocks,

adding said assigned logical processor number to each instruction in each of said plurality of basic blocks in response to said assignment,

forming execution sets (ESs) of basic blocks in response to said steps of adding said instruction firing times and logical processor numbers,

(i) separately storing the instructions contained within a given formed execution set based on the logical processor number, each group of said separately stored instructions containing instructions having only the same logical processor number,

(ii) selectively connecting said separately stored instructions to said processor elements elements based on the logical processor number,

(iii) each said processor element receiving the instruction having the earliest instruction firing time (IFT) first, said processor element having means for performing the steps of:

(a) obtaining input data for processing said received instruction from a shared storage location in said plurality of shared resource locations as identified by said instruction,

(b) storing the results based upon the aforesaid step of processing in a shared storage location in said plurality of shared resource locations as identified by said instruction,

(c) repeating the aforesaid steps (a) and (b) for the next received instruction until all instructions are processed, and

(iv) repeating the aforesaid steps (i) through (iii) for all execution sets that are processed.

26. A system for parallel processing natural concurrencies in a program, said program having a plurality of single entry-singe exit (SESE) basic blocks (BBs) wherein each of said basic blocks contains a stream of instructions, said system comprising:

means (**160**) receptive of said plurality of basic blocks for determining said natural concurrencies within said instruction stream for each of said basic blocks, said determining means further having means for adding timing and processor information to each instruction in response to said determined natural concurrencies so that all processing resources required by any given instruction are allocated in advance of program execution,

means (**620**) receiving said basic blocks (BBs) of instructions having said added timing and processor information for storing said received instructions,

a plurality of processor elements (PEs),

means (**650**) for selectively connecting said plurality of processor elements to said storing means,

a plurality of shared resource (**660**),

means (**670**) for selectively interconnecting said plurality of processor elements (PEs) with said plurality of shared resources (**660**),

said storing means having means for delivering instructions in order of the earliest firing time first, based upon said firing timing information, to the processor elements over said connecting means and into said processor elements, and

said processor elements having means for processing each received instruction from said storing means (**620**), said processor elements being connected to the shared resources identified by each said instruction and wherein all resource information and context information pertaining to said instruction are each stored in one of said plurality of shared resources and said storing means (**620**).

27. The parallel processor system of claim **26** in which:

said determining means further has means for adding program level information to said instructions, said information containing relative level information for each said instruction in order to identify the different program levels associated with registers used within said program,

said shared resources having a different set of registers associated with each program level, and

said processor elements further having means for processing each of its received instructions using at least one set of registers identified by said received instructions.

28. The system of claim **26** wherein said determining means further comprises means for forming execution sets (ESs) from the basic blocks containing said added timing and processor information wherein branches from any given basic block within a given execution set out of said given execution set to a basic block in another execution set is statistically minimized.

29. The system of claim **28** wherein said determining means further has means for attaching header information to each formed execution set, said header at least comprising:

(a) the address of the start of said instructions, and

(b) the length of the execution set.

30. The system of claim **26** wherein said storing means further comprises:

a plurality of caches (**1522**) receptive of said execution sets for storing said instructions,

means (**1544, 1560, 1570, 650**) connected to said caches for delivering said instructions stored in each of said caches to said plurality of processor elements, and

means (**1512, 1518, 1548**) connected to said caches and said delivering means for controlling said storing and said delivery of instructions, said controlling means further having means for executing the branches from individual basic blocks.

31. The system of claim **26** wherein said determining means further comprises means for adding level information to instructions pertaining to the different program levels contained within said program, and wherein each said processor element has means for processing each of its received instructions using a set of shared resources identified by each said instruction's level information.

32. The system of claim **26** wherein each of said plurality of processor elements is context free in processing a given instruction.

33. The system of claim **32** wherein said plurality of shared resources comprises:

a plurality of register files, and

a plurality of condition code files, said plurality of register files and said plurality of condition code files together with said storing means (**620**) having means for storing all necessary context data for processing any given instruction.

34. A system for parallel processing natural concurrencies in a program, said program having a plurality of single entry-single exit (SESE) basic blocks (BBs) wherein each of said basic blocks (BBs) contains a stream of instructions, said system comprising:

means (**160**) receptive of said plurality of basic blocks for determining said natural concurrencies within said instruction stream for each of said basic blocks, said determining means further having means for adding timing, processor, and resource access information to each instruction in response to said determined natural concurrencies so that all processing resources required by any given instruction are allocated in advance of instruction execution, said determining means further having means for forming basic blocks into execution set (ESs) containing said added instruction firing times and logical processor numbers, wherein branches from any given basic block within a given execution set out of said given execution set to a basic block in another execution set is statistically minimized,

means (**620**) receptive of said execution sets having said added information for storing said instructions,

a plurality of context-free processor elements connected to said storing means, and

a plurality of shared resources connected to said plurality of context-free processor elements, said processor elements having means for processing its instructions based upon said processor number, timing, and resource access information from said storing means, said processor elements having means for obtaining all necessary context data from said plurality of shared resources based upon location data set forth in said instructions for processing said instruction and having means for delivering all necessary context data to said plurality of shared resources at locations based upon and identified in said instructions.

35. A parallel processor system for processing streams of low level instructions contained in a plurality of programs in said system, each of said streams having a plurality of single entry-single exit basic blocks, said system comprising:

means (**160**) for adding intelligence to the instruction streams in response to detected natural concurrencies therein,

a plurality of context files, each of said plurality of programs being assigned to one of said context files, each of said context files having a plurality of register resources,

a plurality of logical resource drivers, each logical resource driver being assigned to one of said plurality of context files, each of said logical resource drivers being receptive of said basic blocks of at least one corresponding program instruction stream from said adding means for storing said instructions of each basic block,

a plurality of individual processor elements,

means for selectively connecting any of said plurality of processor elements to any of said plurality of logical resource drivers for transferring said instructions from each of said logical resource drivers to individually assigned processor elements in accordance with said added intelligence in the instructions,

first means for connecting each of said processor elements to any one of said plurality of context files, each said processor element having means for accessing any resource in a program's context file for reading and writing data during said processing of the program's instructions,

a plurality of memory locations (**610**), and

second means for connecting each of said logical resource drivers with any one of said plurality of memory locations, each said logical resource driver having means for accessing said memory locations during said processing of each said instruction.

36. A parallel processor system for processing streams of low level instructions contained in a plurality of programs, each of said streams having a plurality of single-entry-single exit basic blocks, said system comprising:

means for adding intelligence to said instruction streams, said added intelligence containing subroutine level information for said instructions to specify the relative subroutine register accesses contained within each said program,

a plurality of context files, each of said plurality of programs being assigned to one of said context files, each of said context files having a plurality of register resources with one set of register resources for each said subroutine program level, each said set of resources being identified with a different subroutine level,

a plurality of logical resource drivers, each logical resource driver being associated with one of said plurality of context files, each of said logical resource drivers being receptive of said basic blocks corresponding to the program instruction stream of said at least one assigned program from said adding means for storing said instructions of each basic block, each of said logical resource drivers further having means for adding information to each said instruction, said added information to each said instruction, said added information containing the

identity of the context file assigned to the programs contained within each said logical resource driver and, in response to the added intelligence, the level of any subroutine access of the context files by the processor,

a plurality of individual processor elements,

means connecting said plurality of processor elements and said plurality of logical resource drivers for transferring said instructions from each of said logical resource drivers to individually assigned processor elements, and

first means (**670**) for connecting each of said processor elements with any one of said plurality of context files, each said processor element having means for accessing a set of resources, in a program's context file as identified by said added information, during said processing of the program's instruction in the logical resource drivers.

37. A multiprocessor system for processing a plurality of programs of different users, said system comprising:

a plurality of logical resource drivers, each said logical resource driver being operative on at least one of said programs for dynamically adding information during program execution to instructions of said one program, said information identifying at least the user context file for said one program,

a plurality of set of shared resources, each of said sets being assigned to a given context file, and

a plurality of processor elements for processing said programs, said processor elements being connected to said plurality of sets of shared resources and connected to said logical resource drivers for receiving instructions in a determined order, each of said plurality of logical resource drivers having means for delivering an instruction to any processor element, each of said processor elements being selectively interconnected with that set of the shared resources identified by said user context information added to the instruction then being processed in order to access all data necessary for processing said instruction.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 4,847,755
DATED : July 11, 1989
INVENTOR(S) : Gordon E. Morrison et al

It is certified that error appears in the above—identified patent and that said Letters Patent is hereby corrected as shown below:

In the title, "Throughout" should be --Throughput--

In column 2, at line 65, "Gajewski" should be --Gajski--

In column 18, at line 31, "PELRD" should be --PE-LRD--

In column 18, at line 46, "finegrained" should be --fine-grained--

In column 33, at line 26, "f" should be --of--

In column 46, at line 43, "to" should be --two--

In column 48, at line 40 (claim 2), "concurrence" should be --concurrencies--

In column 51, at line 38 (claim 7), after "programs" insert --,--

In column 54, at line 39 (claim 13), after "said" insert --procesor elements having means for accessing any one of said--

In column 55, at line 19 (claim 15), "instruction" should be --instructions--

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 4,847,755

DATED : July 11, 1989

INVENTOR(S) : Gordon E. Morrison et al

It is certified that error appears in the above—identified patent and that said Letters Patent is hereby corrected as shown below:

In column 56, at line 48 (claim 19), "storage" should be --storing--

In column 57, at line 6 (claim 20), "of" should be --or--

In column 57, at line 51 (claim 21), after "of" insert --said--

In column 58, at line 14 (claim 22), "element" should be --elements--

In column 58, at line 27 (claim 22), "of" should be --or--

In column 59, at line 21 (claim 25), before "instructions" delete "method"

In column 59, at line 52 (claim 25), delete "elements" (second occurrence)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 4,847,755

DATED : July 11, 1989

INVENTOR(S) : Gordon E. Morrison, et al

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

In column 62, at lines 67-68 (claim 36), after "information" delete "to each said instruction, said added information"

In column 64, line 7 (claim 37), "set" should be --sets--

Signed and Sealed this

Twenty-sixth Day of February, 1991

Attest:

HARRY F. MANBECK, JR.

Attesting Officer

Commissioner of Patents and Trademarks